Isomorphisms of types in the presence of higher-order references

Pierre Clairambault
University of Bath
p.clairambault@bath.ac.uk

***Abstract*—We investigate the problem of type isomorphisms in a programming language with higher-order references. We first recall the game-theoretic model of higher-order references by Abramsky, Honda and McCusker. Solving an open problem by Laurent, we show that two finitely branching arenas are isomorphic if and only if they are geometrically the same, up to renaming of moves (Laurent's forest isomorphism). We deduce from this an equational theory characterizing isomorphisms of types in a finitary language $\mathcal{L}_2$ with higher order references. We show however that Laurent's conjecture does not hold on infinitely branching arenas, yielding a non-trivial type isomorphism in the extension of $\mathcal{L}_2$ with natural numbers.**

I. Introduction

During the development of denotational semantics of programming languages, there was a crucial interest in defining models of computation satisfying particular type equations. For instance, a model of the untyped λ-calculus can be obtained by isolating a *reflexive* object (that is, an object D such that $D \simeq D^D$) in a cartesian closed category. In the 80s, some people started to consider the dual problem of finding these equations that must hold in *every* model of a given language: they were coined *type isomorphisms* by Bruce and Longo. In [8], they exploited a theorem by Dezani [9] giving a syntactic characterization of invertible terms in the untyped λ-calculus to prove that that the only isomorphisms of types present in simply typed λ-calculus with respect to $\beta\eta$ equality are those induced by the equation $A \to (B \to C) \simeq B \to (A \to C)$. Later this was extended to handle such things as products [7], higher order [8], possibly with unit types [10], or sums [11].

The interest in type isomorphisms grew significantly when their practical impact was realized. In [23], Rittri proposed to search functions in software libraries using their type modulo isomorphism as a key. He also considered the possibilities offered by matching and unification of types modulo isomorphisms [24]. A whole line of research has also been dedicated to the study of type isomorphisms and their use for search tools in richer type systems (such as dependent types [5]), along with studies about the automatic generation of the corresponding coercions [4]. Such tools were implemented for several programming languages, let us mention the command line tool `camlsearch` written by Vouillon for CamlLight.

It is worth noting that even though these tools are written for powerful programming languages featuring complex computational effects such as higher-order references or exceptions, they rely on the theory of isomorphisms in weaker (purely functional) languages, such as the second-order λ-calculus with pairs and unit types for `camlsearch`. Clearly, all type isomorphisms in λ-calculus are still valid in the presence of computational effects (indeed, the operational semantics are compatible with $\beta\eta$). What is less clear is whether those effects allow the definition of new isomorphisms. However, it seems that syntactic methods deriving from Dezani's theorem on invertible terms in λ-calculus cannot be extended to complex computational effects. The base setting itself is completely different: the dynamics of terms are no longer defined by reduction rules but by operational semantics, the natural equality between terms is no longer convertibility but observational equivalence, so new methods are required.

In [18], Laurent introduced the idea of applying game semantics to the study of type isomorphisms (although one should mention the precursor characterization of isomorphisms by Berry and Curien [6] in the category of concrete data structures and sequential algorithms). Exploiting his earlier work on game semantics for polarized linear logic [17], he found the theory of isomorphisms for LLP from which he deduced (by translations) the isomorphisms for the call-by-name and call-by-value $\lambda\mu$-calculus. The core of his analysis is the observation that isomorphisms between arenas A and B in the category $\mathbf{Inn}$ [13] of arenas and innocent strategies are in one-to-one correspondence with *forest isomorphisms* between A and B, so in particular two arenas are isomorphic if and only if their representations as forests are identical up to the renaming of vertices.

From the point of view of computational effects this looks promising, since game semantics are known to accommodate several computational effects such as control operators [15], ground type [2] or higher-order references [1] or even concurrency [16] in one single framework. Moreover, Laurent pointed out in [18] that the main part of his result, namely the fact that each $\mathbf{Inn}$-isomorphism induces a forest isomorphism, does not really depend on the innocence hypothesis but only on the weaker *visibility* condition. As a consequence, his method for characterizing isomorphisms still applies to programming languages such as Idealized Algol whose terms can be interpreted as visible strategies [2]. Laurent raised the question whether his result could be proved without the visibility condition, therefore yielding a characterization of isomorphisms in a programming language whose terms have access to higher-order references and hence get interpreted as non-visible strategies [1].

The contribution of this paper is threefold: *(1)* We give a new and synthetic reformulation of Laurent's tools to approach

$$\begin{array}{rcl} A \times B & \simeq_{\mathcal{E}} & B \times A \\ A \times (B \times C) & \simeq_{\mathcal{E}} & (A \times B) \times C \\ A \times \mathtt{unit} & \simeq_{\mathcal{E}} & A \\ \mathtt{bool} \times A \to B & \simeq_{\mathcal{E}} & (A \to B) \times (A \to B) \\ \mathtt{var}[A] & \simeq_{\mathcal{E}} & (A \to \mathtt{unit}) \times (\mathtt{unit} \to A) \end{array}$$

Figure 1. Isomorphisms in $\mathcal{L}_2$

game-theoretically the problem of type isomorphisms, *(2)* We prove Laurent's conjecture in the case of finitely branching arenas, allowing us to characterize all type isomorphisms in a finitary (integers-free) programming language $\mathcal{L}_2$ with higher-order references by the theory $\mathcal{E}$ presented[1] in Figure 1, *(3)* We show however a counter-example to the conjecture when dealing with infinitely branching arenas, and the counter-example yields a non-trivial type isomorphism between the types $(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{unit}) \to \mathtt{unit}$ and $(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{unit}) \to \mathtt{unit}$ in the extension of $\mathcal{L}_2$ with natural numbers. So Laurent's conjecture, in the general case, is false.

In Section II we introduce the finitary language $\mathcal{L}_2$ strongly inspired by Abramsky, Honda and McCusker's language $\mathcal{L}$ [1], along with its standard game semantics. Then we turn to the problem of isomorphisms of types. In Section III we first give an analysis of isomorphisms in several subcategories of the games model, reproving and extending Laurent's theorem, then we use it to characterize isomorphisms in $\mathcal{L}_2$. We show how this characterization fails in the presence of natural numbers, and we give a non-trivial type isomorphism in $\mathcal{L}$.

II. The language $\mathcal{L}_2$ and its game semantics

A. Definition of $\mathcal{L}_2$

a) Basic definitions: We introduce here a finitary variant $\mathcal{L}_2$ of the programming language $\mathcal{L}$ with higher-order references modeled by Abramsky, Honda and McCusker in [1]: it only differs from $\mathcal{L}$ in the fact that the type of natural numbers has been replaced with a type for booleans, along with all the associated combinators. The terms and types of $\mathcal{L}_2$ are given by the following grammars.

$$A ::= \mathtt{unit} \mid \mathtt{bool} \mid A \times A \mid A \to A \mid \mathtt{var}[A]$$

$$\begin{array}{rcl} M & ::= & x \mid \lambda x.M \mid M\ M \mid \langle M, M \rangle \mid \mathtt{fst}\ M \mid \mathtt{snd}\ M \\ & & \mid \mathtt{skip} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{if}\ M\ M\ M \\ & & \mid \mathtt{new}_A \mid M := M \mid !M \mid \mathtt{mkvar}\ M\ M \end{array}$$

The typing rules for λ-calculus, pairs and booleans are standard. The rules for references follow.

$$\frac{}{\Gamma \vdash \mathtt{new}_A : \mathtt{var}[A]} \qquad \frac{\Gamma \vdash M : \mathtt{var}[A]}{\Gamma \vdash !M : A}$$

$$\frac{\Gamma \vdash M : \mathtt{var}[A] \quad \Gamma \vdash N : A}{\Gamma \vdash M := N : \mathtt{unit}}$$

$$\frac{\Gamma \vdash M : A \to \mathtt{unit} \quad \Gamma \vdash N : \mathtt{unit} \to A}{\Gamma \vdash \mathtt{mkvar}\ M\ N : \mathtt{var}[A]}$$

This language is equipped with a standard big-step call-by-value operational semantics. To define it, we temporarily extend the syntax of terms with identifiers for **locations**, denoted by l. Then, **values** are formed as follows:

$$V ::= \mathtt{skip} \mid \mathtt{true} \mid \mathtt{false} \mid \lambda x.M \mid \langle V, V \rangle \mid l \mid \mathtt{mkvar}\ V\ V$$

The operational semantics of $\mathcal{L}_2$ are then given as an inductively generated relation $(L, s)M \Downarrow (L', s')V$, where s is a partial map from locations in L to values. The rules for λ-calculus, products and booleans are standard (they do not affect the store) and we give in Figure 2 the rules for references. Note that as usual, some store annotations are omitted to aid readability; the rules can be disambiguated as explained in [1]. For a closed term M without free locations, we write $M \Downarrow$ to indicate that $(\emptyset, \emptyset)M \Downarrow (L, s)V$ for some L, s and V. The observational preorder $M \leq N$ between terms M and N is then defined as usual, by requiring that for all contexts $C[-]$ such that $C[M]$ and $C[N]$ are closed and contain no free location, if $C[M] \Downarrow$ then $C[N] \Downarrow$. The corresponding equivalence relation is denoted by $\cong$.

b) Syntactic extensions: In this core language, one can define all the constructs of a basic imperative programming language. For instance if C_1 has type $\mathtt{unit}$, sequential composition $C_1; C_2$ is given by:

$$(\lambda d : \mathtt{unit}.\ C_2)\ C_1$$

This works only because the evaluation of $\mathcal{L}_2$ is call-by-value. Likewise, a variable declaration $\mathtt{new}\ x : A\ \mathtt{in}\ N$ (where M has type A) can be obtained by

$$(\lambda x : \mathtt{var}[A].\ N)\ \mathtt{new}_A$$

and its initialized variant $\mathtt{new}\ x = M\ \mathtt{in}\ N$ as expected. As usual with general references one can define a fixed point combinator Y by

$$\begin{array}{l} \lambda f : (A \to B) \to (A \to B). \\ \quad \mathtt{new}\ y : A \to B\ \mathtt{in} \\ \quad\ y := \lambda a : A.\ f\ !y\ a; \\ \quad\ !y \end{array}$$

This can be easily applied to implement a `while` loop. We can also use it to build an inhabitant $\bot$ to any type A.

c) Bad variables and isomorphisms: The `mkvar` construct allows to combine arbitrary "write" and "read" methods, forming terms of type $\mathtt{var}[A]$ not behaving as reference cells: those are called *bad variables*. We include bad variable in $\mathcal{L}_2$ for two reasons. First, because games models that allow bad variables are notably simpler than those which do not [22], for which it is not clear whether our methods apply. Second, because we expect the problem of isomorphisms without bad variables to be far more subtle than what we consider here, because of the observation by O'Hearn that

[1]The absence of the equation $A \to (B \to C) \simeq B \to (A \to C)$ mentioned in the introduction may seem strange, but is standard in call-by-value [18] due to the restriction of the η-rule on values.

$$\frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2}{\texttt{mkvar}\ M_1\ M_2 \Downarrow \texttt{mkvar}\ V_1\ V_2} \qquad \frac{}{(L,s)\ \texttt{new}_A \Downarrow (L \cup \{l : A\}, s)\ l}\ (l \notin L) \qquad \frac{(L,s)M \Downarrow (L',s')l \quad (L',s')N \Downarrow (L'',s'')V}{(L,s)M := N \Downarrow (L'',s''(l \mapsto V))\texttt{skip}}$$

$$\frac{M \Downarrow \texttt{mkvar}\ V_1\ V_2 \quad N \Downarrow V \quad V_1(V) \Downarrow \texttt{skip}}{M := N \Downarrow \texttt{skip}} \qquad \frac{(L,s)M \Downarrow (L',s')l \quad s'(l) = V}{(L,s)!M \Downarrow (L',s')V} \qquad \frac{M \Downarrow \texttt{mkvar}\ V_1\ V_2 \quad V_2(\texttt{skip}) \Downarrow V}{!M \Downarrow V}$$

Figure 2. Big-step operational semantics for references in $\mathcal{L}_2$.

without bad variables, not only $\texttt{var}[X]$ is not functorial, but it does not even preserve isomorphisms. However, note that var-free isomorphisms are the same with or without bad variables.

d) Isomorphisms of types: We are now ready to define the notion of isomorphism of types in $\mathcal{L}_2$.

DEFINITION 1. *If A and B are two types of $\mathcal{L}_2$, we say that A and B are* isomorphic*, denoted by $A \simeq_{\mathcal{L}_2} B$, if and only if there are two terms $x : A \vdash M : B$ and $y : B \vdash N : A$ such that:*

$$\begin{aligned}(x : A \vdash (\lambda y.N)M) &\cong id_A\\ (y : B \vdash (\lambda x.M)N) &\cong id_B\end{aligned}$$

where $id_A = x : A \vdash x : A$.

B. The games model

We now describe the fully abstract games model of $\mathcal{L}_2$. Note that except a few details there is nothing new here, as this is precisely the model described in [1]. We however include the definitions (but no proofs) for the sake of self-completeness.

e) Arenas, plays: Our games have two participants: Player (P) and Opponent (O). Valid plays between O and P are generated by directed graphs called *arenas*, which are the abstract representation of types. An **arena** is a tuple $A = \langle M_A, \lambda_A, I_A, \vdash_A \rangle$ where

- M_A is a set of **moves**,
- $\lambda_A : M_A \to \{O,P\} \times \{Q,A\}$ is a **labeling** function which indicates whether a move is by Opponent or Player, and whether it is a Question or Answer. We write

$$\begin{aligned}\{O,P\} \times \{Q,A\} &= \{OQ, OA, PQ, PA\}\\ \lambda_A &= \langle \lambda_A^{OP}, \lambda_A^{AQ} \rangle\end{aligned}$$

  The function $\overline{\lambda_A}$ denotes λ_A with the O/P part reversed. A move $a \in M_A$ is a O-move (resp. P-move) if $\lambda_A(a) = O$ (resp. $\lambda_A(a) = P$).
- $I_A \subseteq {\lambda_A}^{-1}(\{OQ\})$ is a set of **initial moves**
- $\vdash_A \subseteq M_A^2$ is a relation called **enabling**, which satisfies that if $a \vdash_A b$, then $\lambda_A^{OP}(a) \neq \lambda_A^{OP}(b)$, and if $\lambda_A^{QA}(b) = A$ then $\lambda_A^{QA}(a) = Q$.

We require two additional conditions on arenas: they should be **complete** (for each question $m \in M_A$, there should be an answer $n \in M_A$ such that $m \vdash_A n$) and **finitely branching** (for all $a \in M_A$, the set $\{m \in M_A \mid a \vdash_A m\}$ is finite). We consider the usual arrow construction $A \Rightarrow B$ on arenas, as well as products $\Pi_{i \in I} A_i$ and lifted sums $\Sigma_{i \in I} A_i$ of finite families of arenas. Their definitions can be found, for example, in [1]. It is obvious that they preserve the fact of being complete and finitely branching. The 0-ary product (the empty arena) is denoted by $\mathbf{1}$, and will be a terminal object in our category.

If A is an arena, a **justified sequence** over A is a sequence of moves in M_A together with **justification pointers**: for each non-initial move b, there is a pointer to an earlier move a such that $a \vdash_A b$. In this case, we say that a **justifies** b. The transitive closure of the justification relation is called **hereditary justification**.

f) Notations: The relation $\sqsubseteq$ will denote the prefix ordering on justified sequences. By $s \sqsubseteq^P t$, we mean that s is a P-ending prefix of t. If s is a sequence, then $|s|$ will denote its length. We also define the **prefix functions** ip and jp by $\mathrm{ip}(\epsilon) = \epsilon$ and $\mathrm{ip}(sa) = s$, and $\mathrm{jp}(si) = \epsilon$ if i is initial, $\mathrm{jp}(s_1 a s_2 b) = s_1 a$ if b is justified by a.

A justified sequence s over A is a **legal play** if it is:

- **Alternating**: If $s'ab \sqsubseteq s$, then $\lambda_A^{OP}(a) \neq \lambda_A^{OP}(b)$.
- **Well-bracketed**: a question q is **answered** by a later answer a if q justifies a. A justified sequence s is well-bracketed if each answer is justified by the last unanswered question, that is, the **pending** question.

The set of all legal plays on A is denoted by $\mathcal{L}_A$. We will also be interested in the set $\mathcal{L}'_A$ of well-bracketed but not necessarily alternating plays on A, called **pre-legal plays**.

g) Strategies, composition: A **strategy** σ on an arena A (denoted $\sigma : A$) is a non-empty set of P-ending legal plays on A satisfying **prefix-closure**, *i.e.* that for all $sab \in \sigma$, we have $s \in \sigma$ and **determinism**, *i.e.* that if $sab, sac \in \sigma$, then $b = c$. As usual, strategies form a category which has arenas as objects, and strategies $\sigma : A \Rightarrow B$ as morphisms from A to B. If $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are strategies, their composition $\sigma;\tau : A \Rightarrow C$ is defined as usual by first defining the set of **interactions** $u \in I(A,B,C)$ of plays $u \in \mathcal{L}_{(A \Rightarrow B) \Rightarrow C}$ such that $u_{\restriction A,B} \in \mathcal{L}_{A \Rightarrow B}$, $u_{\restriction B,C} \in \mathcal{L}_{B \Rightarrow C}$ and $u_{\restriction A,C} \in \mathcal{L}_{A \Rightarrow C}$ (where $s_{\restriction A,B}$ is the usual restriction operation essentially taking the subsequence of s in M_A and M_B, along with the possible natural reassignment of justification pointers). The **parallel interaction** of σ and τ is then the set $\sigma||\tau = \{u \in I(A,B,C) \mid u_{\restriction A,B} \in \sigma \wedge u_{\restriction B,C} \in \tau\}$, and the composition of σ and τ is obtained by the **hiding** operation, *i.e.* $\sigma;\tau = \{u_{\restriction A,C} \mid u \in \sigma||\tau\}$. It is known (e.g. [19]) that composition is associative. It admits *copycat strategies* as identities: $id_A = \{s \in \mathcal{L}_{A_1 \Rightarrow A_2} \mid \forall s' \sqsubseteq^P s, s'_{\restriction A_1} = s'_{\restriction A_2}\}$.

If $s \in \mathcal{L}_A$, the **current thread** of s, denoted $\lceil s \rceil$, is the subsequence of s consisting of all moves hereditarily justified by the same initial move as the last move of s. All strategies we are interested in will be *single-threaded*, *i.e.* they only depend on the current thread. Formally, $\sigma : A$ is **single-threaded** if

- For all $sab \in \sigma$, b points in $\ulcorner sa \urcorner$,
- For all $sab, t \in \sigma$ such that $ta \in \mathcal{L}_A$ and $\ulcorner sa \urcorner = \ulcorner ta \urcorner$, we have $tab \in \sigma$.

It is straightforward to prove that single-threaded strategies are stable under composition and that id_A is single-threaded. Hence, there is a category $\mathbf{Gam}$ of arenas and single-threaded strategies. The category $\mathbf{Gam}$ will be the base setting for our analysis. Given arenas A and B, the arena $A \times B$ defines a cartesian product of A and B and the construction $A \Rightarrow B$ extends to a right adjoint $A \times - \dashv A \Rightarrow -$, hence $\mathbf{Gam}$ is cartesian closed and is a model of simply typed λ-calculus. It also has *weak coproducts*, given by the lifted sum [1].

h) Views, classes of strategies: In this paper, we are mainly interested in the properties of single-threaded strategies. However, to give a complete account of the context it seems necessary to mention several classes of strategies of interest in this setting. The most important one is certainly the class of *innocent* strategies, both for historical reasons and because it is at the core of the frequent definability results – and thus of the full abstraction results – in game semantics. Their definition requires the notion of P-view, defined as usual by induction on plays as follows.

$$\begin{array}{rcll} \ulcorner si \urcorner & = & i & \text{if } i \in I_A \\ \ulcorner sa \urcorner & = & \ulcorner s \urcorner & \text{if } \lambda_A^{OP}(a) = P \\ \ulcorner s_1 a s_2 b \urcorner & = & \ulcorner s_1 \urcorner ab & \text{if } \lambda_A^{OP}(b) = O \text{ and } a \text{ justifies } b \end{array}$$

A strategy $\sigma : A$ is then said to be **visible** if it always points inside its P-view, that is, for all $sab \in \sigma$ the justifier of b appears in $\ulcorner sa \urcorner$. The strategy σ is **innocent** if it is visible, and if its behaviour only depends on the information contained in its P-view. More formally, whenever $sab, t \in \sigma$ such that $ta \in \mathcal{L}_A$ and $\ulcorner sa \urcorner = \ulcorner ta \urcorner$, we must also have $tab \in \sigma$. Both visibility and innocence are stable under composition [13], [2], thus let us denote by $\mathbf{Vis}$ the category of arenas and visible single-threaded strategies and by $\mathbf{Inn}$ the category of arenas and innocent strategies. Both categories inherit the cartesian closed structure of $\mathbf{Gam}$, but strategies in $\mathbf{Inn}$ are actually nothing but abstract representation of (η-long β-normal) λ-terms and form a fully complete model of simply-typed λ-calculus. Strategies in $\mathbf{Vis}$ have more freedom, they correspond in fact to programs with first-order store [2].

C. Modeling $\mathcal{L}_2$ in $\mathrm{Fam_f}(\mathbf{Gam})$

i) Interpretation: The three categories $\mathbf{Gam}$, $\mathbf{Vis}$ and $\mathbf{Inn}$ are categories of *negative games* (in which Opponent always plays first), and these are known to model call-by-name computation whereas $\mathcal{L}_2$ is call-by-value. We could have modeled it using positive games, following the lines of [12]. Instead, we follow [1] and model $\mathcal{L}_2$ in the free completion $\mathrm{Fam}(\mathbf{Gam})$ of $\mathbf{Gam}$ with respect to coproducts. This will allow us to first characterize isomorphisms in $\mathbf{Gam}$ (result which could be applied to a call-by-name language with state) then deduce from it the isomorphisms in $\mathrm{Fam}(\mathbf{Gam})$. In fact we will consider the completion $\mathrm{Fam_f}(\mathbf{Gam})$ of $\mathbf{Gam}$ with respect to *finite* coproducts, since $\mathcal{L}_2$ has only finite types.

The objects of $\mathrm{Fam_f}(\mathbf{Gam})$ are finite families $\{A_i \mid i \in I\}$ of arenas. A map from $\{A_i \mid i \in I\}$ to $\{B_j \mid j \in J\}$ is the data of a function $f : I \to J$ together with a family of strategies $\{\sigma_i : A_i \to B_{f(i)} \mid i \in I\}$. As shown in [3], the cartesian closed structure of $\mathbf{Gam}$ extends to $\mathrm{Fam_f}(\mathbf{Gam})$. Moreover, the weak coproducts in $\mathbf{Gam}$ give rise to a *strong monad* T on $\mathrm{Fam_f}(\mathbf{Gam})$. Given families $A = \{A_i \mid i \in I\}$ and $B = \{B_j \mid j \in J\}$, we define

$$\begin{array}{rcl} A \times B & = & \{A_i \times B_j \mid (i,j) \in I \times J\} \\ A \Rightarrow B & = & \{\Pi_{i \in I}(A_i \Rightarrow B_{f(i)}) \mid f : I \to J\} \\ TA & = & \{\Sigma_{i \in I} A_i\} \end{array}$$

The singleton family $\{\mathbf{1}\}$ is the terminal object of $\mathrm{Fam_f}(\mathbf{Gam})$. By abuse of notation, we will still denote it by $\mathbf{1}$. The other components of the cartesian closed structure of $\mathrm{Fam_f}(\mathbf{Gam})$ and of the strong monad structure of T follow naturally from these definitions. We skip the details, as all of this is already covered in [1]. It is known that given a cartesian closed category with a strong monad, one can interpret call-by-value languages in the Kleisli category of the monad [21], and the interpretation of $\mathcal{L}_2$ in $\mathrm{Fam_f}(\mathbf{Gam})$ follows these lines.

We interpret `unit` and `bool` as the families with respectively one and two elements whose components are all empty arenas. Of course we also need to give an interpretation for $\mathtt{var}[A]$, along with morphisms for the read and write operations of the reference cell. Once again, we follow the lines of [1] and consider the type $\mathtt{var}[A]$ as the product of its read and write methods, hence we set $[\![\mathtt{var}[A]]\!] = ([\![A]\!] \Rightarrow T\mathbf{1}) \times T[\![A]\!]$. The interpretation relies on the definition of a morphism $\mathbf{1} \to [\![\mathtt{var}[A]]\!]$, that is, if $[\![A]\!] = \{A_i \mid i \in I\}$, a strategy $\mathtt{cell} : (\Pi_{i \in I}(A_i \Rightarrow \mathbf{1}_\perp) \times \Sigma_{i \in I} A_i)_\perp$, where $A_\perp = T\{A\}$ is the lift operation. Apart from the initial protocol due to the lift, the strategy `cell` works by associating each read request with the latest write request and playing copycat between them. A more detailed description is given in [1], and an algebraic definition is obtained in [20]. As proved in [1], this gives a sound interpretation of $\mathcal{L}_2$ in $\mathrm{Fam_f}(\mathbf{Gam})_T$.

j) Complete plays and full abstraction: A fully abstract model of $\mathcal{L}_2$ is obtained by quotienting $\mathrm{Fam_f}(\mathbf{Gam})_T$ by the usual observational preorder. However, as is often the case with fully abstract game semantics of languages with store, it is *effectively presentable*: the observational preorder can be characterized directly. Say a play $s \in \mathcal{L}_A$ is **complete** if it has as many questions and answers, *i.e.* all questions have been answered. If σ is a strategy on an arena A, then let us denote by $\mathrm{comp}(\sigma)$ the set of complete plays in σ. Take $\sigma, \tau : A \to B$ two morphisms in $\mathrm{Fam_f}(\mathbf{Gam})_T$, with $A = \{A_i \mid i \in I\}$ and $B = \{B_j \mid j \in J\}$. Then, σ and τ consist of families $\{\sigma_i : A_i \to \Sigma_{j \in J} B_j \mid i \in I\}$ and $\{\tau_i : A_i \to \Sigma_{j \in J} B_j \mid i \in I\}$. We then say that $\sigma \preccurlyeq \tau$ if and only if for all $i \in I$, $\mathrm{comp}(\sigma_i) \subseteq \mathrm{comp}(\tau_i)$.

Take now two terms M and N of type A, and suppose $[\![A]\!] = \{A_i \mid i \in I\}$. Then $[\![M]\!]$ and $[\![N]\!]$ are morphisms from $\mathbf{1}$ to $T[\![A]\!]$ in $\mathrm{Fam_f}(\mathbf{Gam})$, *i.e.* strategies on $\Sigma_{i \in I} A_i$. Given the full abstraction result of [1], it is then straightforward to prove the following equivalence:

$$M \leq N \Leftrightarrow [\![M]\!] \preccurlyeq [\![N]\!]$$

This concrete representation of the observational preorder will be central to our characterization of isomorphisms in $\mathcal{L}_2$.

III. Isomorphisms in $\mathbf{Gam}$

We are now going to extend Laurent's tools [18] to characterize isomorphisms of types for $\mathcal{L}_2$. We will first recall Laurent's work in the visible and innocent cases, then extend it to characterize isomorphisms in $\mathbf{Gam}$. From there, we will switch to call-by-value and study the isomorphisms in $\mathrm{Fam_f}(\mathbf{Gam})$, and in its Kleisli category over T.

A. Isomorphisms and zig-zag strategies

We start by giving the definition of (a subtle adaptation of) Laurent's *zig-zag* plays.

Definition 2. *Let $s \in \mathcal{L}_{A \Rightarrow B}$ be a legal play. It is **zig-zag** if*

1. *Each P-move following an O-move in A (resp. in B) is in B (resp. in A),*
2. *A P-move in A immediately follows an initial O-move in B if and only if it is justified by it,*
3. *The (not necessarily legal) sequences $s_{\restriction A}$ and $s_{\restriction B}$ have the same pointers.*

*If s only satisfies the first two conditions, then it is **pre-zig-zag**.*

By extension, we will say that a strategy σ is **pre-zig-zag** (resp. **zig-zag**) if all its plays are so. The core of Laurent's theorem is then that all isomorphisms in $\mathbf{Vis}$ are zig-zag strategies. His proof does rely on visibility, however it only gets involved to prove that the condition 3 of zig-zag plays is satisfied. The first half of his argument does not use visibility and actually proves that all isomorphisms in $\mathbf{Gam}$ are pre-zig-zag. Here, being mainly interested in $\mathbf{Gam}$, we make this explicit. We need first the following lemma.

Lemma 1 (Dual pre-zig-zag play)**.** *Let $s \in \mathcal{L}_{A \Rightarrow B}$ be a pre-zig-zag play, then there exists an unique pre-zig-zag $\overline{s} \in \mathcal{L}_{B \Rightarrow A}$ such that $\overline{s}_{\restriction A} = s_{\restriction A}$ and $\overline{s}_{\restriction B} = s_{\restriction B}$.*

Proof: We define $\overline{s}$ by induction on s; $\overline{\epsilon} = \epsilon$, and $\overline{sab} = \overline{s}ba$. We keep the same pointers, except for the case where a move a in A was justified by an initial move b in B. Then because of the pre-zig-zag condition on s, a is necessarily an initial move in A and is set as the new justifier of b in $\overline{s}$. There is no other possible $\overline{s}$, since the restrictions on A and B are constrained by the hypotheses and their interleaving is forced by the alternation and the pre-zig-zag conditions on $\overline{s}$. ∎

Lemma 2. *If $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow A$ form an isomorphism in $\mathbf{Gam}$, then they are pre-zig-zag and for all s, $s \in \sigma \Leftrightarrow \overline{s} \in \tau$.*

Proof: Consider an isomorphism $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow A$ in $\mathbf{Gam}$. We will prove by induction on even $k \in \mathbb{N}$ that all plays of σ, τ whose length is less than k are pre-zig-zag, and that moreover $\{\overline{s} \mid s \in \sigma \wedge |s| \leq k\} = \{s \in \tau \mid |s| \leq k\}$.

If $k = 0$, this is trivial. Otherwise, suppose this is true up to $k \in \mathbb{N}$, and consider $sab \in \sigma$ of length $k+2$; let us first prove condition (1). Without loss of generality, suppose $a \in M_A$. Since $s_{\restriction B} = \overline{s}_{\restriction B}$, by a straightforward zipping argument we can build an interaction $u \in I(A_1, B, A_2)$ such that $u_{\restriction A_1, B} = s$ and $u_{\restriction B, A_2} = \overline{s}$, moreover since σ, τ form an isomorphism we must have $u_{\restriction A_1, A_2} \in id_A$. Now, we necessarily have $b \in M_B$, otherwise u could be extended to $uab \in \sigma || \tau$ with $uab_{\restriction A_1, A_2} = u_{\restriction A_1, A_2} ab$ which is not a play of the identity, contradiction. Hence sab satisfies condition 1 of pre-zig-zag plays.

To see why it satisfies condition 2, take $sba \in \sigma$ with b in B and a in A. If b is initial in B, then a necessarily points to it since σ is single-threaded. Reciprocally, suppose a points to an initial move in B earlier than b. Then we have $\overline{s} \in \tau$, and by the same zipping argument as above we have an unique $u \in I(B_1, A, B_2)$ such that $u_{\restriction B_1, A} = \overline{s}$ and $u_{\restriction A, B_2} = s$. Since σ, τ form an isomorphism we also have $u_{\restriction B_1, B_2} \in id_B$. Let us now extend u to $u' = ub_2 ab_1$ in the unique way such that $u'_{\restriction A, B_2} = sba$ and $u'_{\restriction B_1, A} \in \tau$. Note that we are sure that b_1 is a move on B_1 since $\overline{s}ab_1$ is a play of τ of length $k+2$ and we already know that these satisfy the condition 1 of pre-zig-zag plays. But we also have $u'_{\restriction B_1, B_2} \in id_B$, hence b_2 points in $\overline{s}$ as b_1 points in s. This means that we have $\overline{s}ab \in \tau$, such that a is initial and b points in $\overline{s}$, impossible since τ is single-threaded. Hence sba satisfies condition 2 of pre-zig-zag plays.

We have proved that sab is pre-zig-zag, so $\overline{sab}$ is defined. By induction hypothesis $\overline{s} \in \tau$ and the same reasoning as above shows that it extends to $\overline{sab} \in \tau$. The argument is symmetric, hence $\{\overline{s} \mid s \in \sigma \wedge |s| \leq k+2\} = \{s \in \tau \mid |s| \leq k+2\}$. ∎

For the sake of completeness, let us include Laurent's argument which proves that isomorphisms in $\mathbf{Vis}$ are zig-zag.

Lemma 3. *If $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow A$ form an isomorphism in $\mathbf{Vis}$, then σ and τ are zig-zag strategies.*

Proof: We already know that σ and τ are pre-zig-zag strategies. We show by induction on $n \in \mathbb{N}$ that for all $s \in \sigma$, if $|s| \leq n$ then $s_{\restriction A}$ and $s_{\restriction B}$ have the same pointers. Take now $s \in \sigma$, and $sab \in \sigma$, suppose w.l.o.g. that $a \in M_A$. Suppose a points to $(s_{\restriction A})_i$, then b points to $(s_{\restriction B})_i$. Indeed, it cannot point to $(s_{\restriction B})_j$ with $j > i$ since that would break visibility for σ. But if it points to $(s_{\restriction B})_j$ with $j \leq i$ we use the same reasoning on the dual pre-zig-zag play $\overline{sab}$ and get a contradiction with the fact that τ is visible. ∎

Let us denote by $\mathbf{Gam}_i$, $\mathbf{Vis}_i$ and $\mathbf{Inn}_i$ the groupoids of arenas and isomorphisms on the respective categories. In the next sections, we use these facts to give more combinatorial representations of $\mathbf{Gam}_i$, $\mathbf{Vis}_i$ and $\mathbf{Inn}_i$.

B. Notions of game morphisms

Laurent's isomorphism theorem works by relating isomorphisms in $\mathbf{Gam}$ with isomorphisms in a simpler category which has arenas as objects and *forest morphisms*[2], *i.e.* maps on moves that preserve initiality and enabling. Relaxing the visibility conditions requires us to also consider relaxed notions of game morphisms, that we present here.

Definition 3. *Let A be an arena. A **path** on A is a play $s \in \mathcal{L}_A$ such that except for the initial move, every move in s points to the previous move. Formally, for all $s'ab \sqsubseteq s$, a justifies b in s. Let $\mathcal{P}_A$ denote the set of paths on A. A **path morphism** from A to B is a function $\phi : \mathcal{P}_A \to \mathcal{P}_B$ such that $\mathrm{ip} \circ \phi = \phi \circ \mathrm{ip}$ and which preserves Q/A labeling: for all*

[2]Note that in [18] arenas are forests, which is not the case here.

$sa \in \mathcal{P}_A$ with $\phi(sa) = \phi(s)b$, we have $\lambda_A^{QA}(a) = \lambda_B^{QA}(b)$. There is a category $\mathbf{Path}$ *of arenas and path morphisms.*

This category $\mathbf{Path}$ comes with its own notion of isomorphisms of arenas. Note that whenever A is a forest, this is exactly Laurent's notion of forest isomorphism. We now introduce two weaker notions of morphisms for arenas. In what follows, let us call a legal play on A with only one initial move a **thread** on A, and denote the set of threads on A by $\mathcal{T}_A$. Likewise, let us call a pre-legal play with one initial move a pre-legal thread and let us denote these by $\mathcal{T}'_A$.

DEFINITION 4. *Let A, B be arenas, and let $\phi : \mathcal{T}'_A \to \mathcal{T}'_B$ We say that ϕ is a* ***sequential morphism*** *from A to B if* $\mathrm{ip} \circ \phi = \phi \circ \mathrm{ip}$*, and if it preserves Q/A labeling,* i.e. *for all $\phi(sa) = \phi(s)b$ we have $\lambda_A^{QA}(a) = \lambda_B^{QA}(b)$. We say that it is a* ***justified morphism*** *if, additionally,* $\mathrm{jp} \circ \phi = \phi \circ \mathrm{jp}$*. There are two categories* $\mathbf{Seq}$ *of arenas and sequential morphisms and* $\mathbf{Jus}$ *of arenas and justified morphisms.*

As above, we will denote by $\mathbf{Seq}_i$, $\mathbf{Jus}_i$ and $\mathbf{Path}_i$ the groupoids of invertible maps in $\mathbf{Seq}$, $\mathbf{Jus}$ and $\mathbf{Path}$. These groupoids will soon appear to be identical to $\mathbf{Gam}_i$, $\mathbf{Vis}_i$ and $\mathbf{Inn}_i$. To prove this, we need the following lemma.

LEMMA 4. *Let $s \in \mathcal{T}'_A$, and $\sigma : A \Rightarrow B$ an isomorphism in* $\mathbf{Gam}$*. There is then an unique play $s' \in \sigma$ such that $s'_{\restriction A} = s$.*

Proof: Remark first that if $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow A$ are inverses then they are both total, *i.e.* for all $s \in \sigma$ and $sa \in \mathcal{L}_{A \Rightarrow B}$ there must be b such that $sab \in \sigma$, assuming it is not the case easily leads to a contradiction. We now prove the lemma by induction on s. If $s = \epsilon$, this is trivial. Otherwise, suppose $sa \in \mathcal{T}'_A$ and we have by induction hypothesis $s' \in \sigma$ such that $s'_{\restriction A} = s$. If a is a P-move in A (hence an O-move in $A \Rightarrow B$), there is an unique b such that $s'ab \in \sigma$, and we do have $s'ab_{\restriction A} = sa$. If a is an O-move in A (hence a P-move in $A \Rightarrow B$), then let $\tau : B \Rightarrow A$ be the inverse of σ, since $s' \in \sigma$ we have $\overline{s'} \in \tau$. Being part of an isomorphism τ is total, hence there is b such that $\overline{s'}ab \in \tau$. We deduce from this that $s'ba \in \sigma$, and we have $s'ba_{\restriction A} = sa$ as needed. This choice is unique: if there is another play $t \in \sigma$ such that $t_{\restriction A} = sa$, then $t = t'b'a$ (since t is zig-zag). By induction hypothesis we have $t' = s'$, thus $s'b'a \in \sigma$. From this we deduce that $\overline{s'}ab' \in \tau$, so $b = b'$ by determinism of τ. ∎

PROPOSITION 1. *If $\mathcal{C} \simeq \mathcal{D}$ means that two groupoids $\mathcal{C}$ and $\mathcal{D}$ are* isomorphic*, then we have:*

$$\begin{aligned} \mathbf{Gam}_i &\simeq \mathbf{Seq}_i \\ \mathbf{Vis}_i &\simeq \mathbf{Jus}_i \end{aligned}$$

Proof: Let us first define a functor $F : \mathbf{Gam}_i \to \mathbf{Seq}_i$. It is defined as the identity on arenas. Let $\sigma : A \Rightarrow B$ be an isomorphism, and let $s \in \mathcal{T}'_A$ then we define $\phi_\sigma(s) = s'_{\restriction B}$, where s' is the unique play on $A \Rightarrow B$ which existence is ensured by the lemma above. The function ϕ_σ commutes with ip since σ is a pre-zig-zag strategy. To any question it cannot associate an answer, as that would immediately break well-bracketing on σ. But to any answer it cannot associate a question, as that would immediately break well-bracketing on σ^{-1}. Then we define $F(\sigma) = \phi_\sigma$. It is obvious that F preserves identities and composition[3].

Reciprocally, suppose $\phi : A \to B$ is a sequential isomorphism. We mimic the usual definition of the identity by setting $G(\phi) = \{s \in \mathcal{L}_{A \Rightarrow B} \mid \forall s' \sqsubseteq^P s,\ \phi(s'_{\restriction A}) = s'_{\restriction B}\}$ (We apply ϕ on plays whereas it is normally only defined on *threads*, however it can be canonically extended to plays, so this is not ambiguous). It is obvious that this construction is functorial, and that it is inverse to F.

We have now an isomorphism $\mathbf{Gam}_i \simeq \mathbf{Seq}_i$ which restricts naturally to $\mathbf{Vis}_i$ and $\mathbf{Jus}_i$. Indeed if $\sigma : A \Rightarrow B$ is a visible isomorphism, it is a zig-zag strategy therefore $s \in \mathcal{T}'_A$ and $\phi_\sigma(s)$ have the same pointers, which means that $\mathrm{jp} \circ \phi_\sigma = \phi_\sigma \circ \mathrm{jp}$. Reciprocally if ϕ_σ is a justified morphism, all $s \in \sigma$ must be such that $s_{\restriction A}$ and $s_{\restriction B}$ have the same pointers, therefore σ, being pre-zig-zag, always points in its P-view. ∎

C. Innocent and visible case

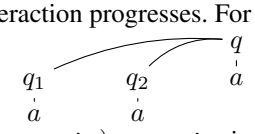
In this section, we use the framework described above to recall Laurent's results. We have proved above that isomorphisms in $\mathbf{Vis}$ correspond to isomorphisms in $\mathbf{Jus}$, which we are now going to compare with isomorphisms in $\mathbf{Path}$.

LEMMA 5. *There is a full functor $H : \mathbf{Vis}_i \to \mathbf{Path}_i$.*

Proof: We have built in the above section a full and faithful functor (actually an isomorphism) $F : \mathbf{Vis}_i \to \mathbf{Jus}_i$. From a visible isomorphism $\sigma : A \Rightarrow B$ we set $H(\sigma) = F(\sigma) \restriction \mathcal{P}_A$, where $f \restriction E'$ restricts a function $f : E \to F$ to a subset $E' \subseteq E$ of its domain. The image of a path by $F(\sigma)$ is always a path since it is a justified morphism, hence $H(\sigma) : \mathcal{P}_A \to \mathcal{P}_B$.

To see why H is full, suppose we have a path morphism $\phi : \mathcal{P}_A \to \mathcal{P}_B$. Then ϕ admits a canonical extension $\phi^* : \mathcal{T}'_A \to \mathcal{T}'_B$. To define $\phi^*(s)$ we reason by induction on s, and set $\phi^*(\epsilon) = \epsilon$ and $\phi^*(sa) = \phi^*(s)a'$, where a' is the last move of $\phi(p_a)$, p_a being the path of a in s. The move a' keeps the same pointer as a. It is clear that this defines as needed a justified morphism ϕ^* such that $H(\phi^*) = \phi$. ∎

This ensures that arenas A and B are isomorphic in $\mathbf{Vis}$ if and only if they are isomorphic in $\mathbf{Path}$, *i.e.* they are geometrically the same. Let us mention that as Laurent proved, this correspondence is one-to-one in the innocent case: one can prove that there is only one innocent zig-zag strategy corresponding to a particular path isomorphism, hence H restricts to an isomorphism of groupoids $H' : \mathbf{Inn}_i \to \mathbf{Path}_i$.

k) Faithfulness of H: Note however that H itself is *not* faithful, because we can exploit non-innocence to build non-uniform isomorphisms, *i.e.* isomorphisms which change their underlying path isomorphism as the interaction progresses. For an example, consider the arena $A =$ 

which is the interpretation of $(\mathtt{bool} \to \mathtt{unit}) \to \mathtt{unit}$ in call-by-value and of $\mathtt{unit} \times \mathtt{unit} \to \mathtt{unit}$ in call-by-name.

[3] In fact, this construction can be seen as a particular case of Hyland and Schalk's faithful functor from games to relations [14], where the relation happens to be functional.

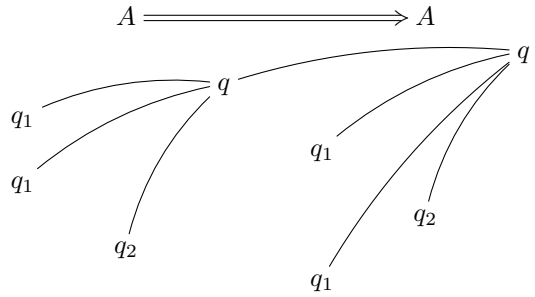


Figure 3. A play of the non-trivial involution i on A

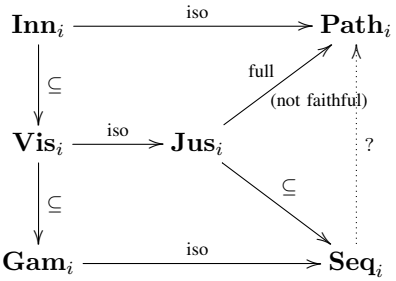


Figure 4. Relations between all groupoids of isomorphisms

Consider now the strategy $i : A \Rightarrow A$ which behaves as follows. It starts by playing as the identity on A. The first time Opponent plays q_1 or q_2 on the left hand side, it simply copies it. Starting from the second time Opponent plays q_1 or q_2 though, it swaps them. An example play of i is given in Figure 3. Although it is not the identity, i is its own inverse. Its image by H only takes into account the first behaviour or i, thus is the same as for id_A: the identity path morphism on A. From this strategy we can extract the following term $f : B \vdash M : B$ of $\mathcal{L}_2$, where $B = (\mathtt{bool} \to \mathtt{unit}) \to \mathtt{unit}$.

$$\mathtt{new}\ r := \mathtt{true}\ \mathtt{in}$$
$$\lambda g.f(\lambda b.\mathtt{if}\ !r\ \mathtt{then}\ r := \mathtt{false}; g\ b\ \mathtt{else}\ g\ (\mathtt{not}\ b))$$

Although M is not the identity it is an involution on B, *i.e.* we have $(\lambda f.M)(Mx) \cong_{\mathcal{L}_2} x$. Such non-trivial involutions cannot be defined using only purely functional behaviour.

We give in Figure 4 a summary of all the groupoids of isomorphisms encountered for the moment, along with their relations. Following it, the question of finding the isomorphisms in $\mathbf{Gam}$ boils down to the definition of an arrow from $\mathbf{Seq}_i$ to $\mathbf{Path}_i$ in this diagram, which is what we will attempt in the next two subsections.

D. Non-visible isomorphisms by counting

We have seen above that we can build a full functor $\mathbf{Vis}_i \to \mathbf{Path}_i$, which allows to characterize isomorphic arenas in $\mathbf{Vis}$. However, this construction relies heavily on visibility. We now investigate how to get rid of it and prove that two arenas A and B are isomorphic in $\mathbf{Gam}$ if and only if they are isomorphic in $\mathbf{Path}$. In this subsection, we will describe for pedagogical reasons an intuitive approach to the proof, which relies on counting. However this approach suffers from some defects, hence the full proof (described in the next subsection) will follow slightly different lines.

If $a \in M_A$, let us call its **arity** the quantity $ar(a) = |\{m \in M_A \mid a \vdash_A m\}|$. On pre-threads $s \in \mathcal{T}'_A$ we define:

$$Q(s) = \sum_{i=1}^{|s|} ar(s_i)$$

If $s \in \mathcal{T}'_A$, $Q(s)$ is also the number of ways s can be extended to some sa (let us recall here that as a member of $\mathcal{T}'_A$, s need not be alternating): the choice of a justifier s_i plus a move enabled by s_i. These definitions allow to express the following observation. If $\sigma : A \Rightarrow B$ is an isomorphism (thus a pre-zig-zag strategy) and $s \in \sigma$, then $Q(s_{\restriction A}) = Q(s_{\restriction B})$, because σ being an isomorphism, it must associate each possible extension of $s_{\restriction A}$ to an unique extension of $s_{\restriction B}$. But this also means that if $sab \in \sigma$ we have $Q(s_{\restriction A}) + ar(a) = Q(s_{\restriction A}a) = Q(s_{\restriction B}b) = Q(s_{\restriction B}) + ar(b)$, hence $ar(a) = ar(b)$. Thus to each move a, σ must associate a move with the same arity. This is a step in the right direction, but we would like a deeper connection between a and b.

If $a \in M_A$, we will use the notation $J_a = \{m \in M_A \mid a \vdash_A m\}$. Let us define by induction on k the notion of a k-isomorphism between $a \in M_A$ and $b \in M_B$. For any $a \in M_A$ and $b \in M_B$ there is automatically a 0-isomorphism $i_{a,b}$. A $(k+1)$-isomorphism from a to b is the data of an isomorphism $f : J_a \to J_b$ along with, for all $m \in J_a$, a k-isomorphism $f_m : m \to f(m)$. We use the notation $m \simeq_k n$ to denote the fact that there is a k-isomorphism from m to n. In other words, we have $m \simeq_k n$ if the tree of paths of length at most k starting form m is tree-isomorphic to the tree of paths of length at most k starting from n. If $k_1 \leq k_2$, f_1 is a k_1-isomorphism and f_2 is a k_2-isomorphism, we say that f_1 is a prefix of f_2 if they agree up to depth k_1. Note that in particular we have $m \simeq_1 n$ if and only if $ar(m) = ar(n)$, so $m \simeq_k n$ is indeed a generalization of $ar(m) = ar(n)$. By induction on k, one can then prove that σ must always associate to each move m a move n such that $m \simeq_k n$: to prove it for $k+1$, just apply the counting argument above on $\simeq_k$-equivalence classes. From all these k-isomorphisms, one can then deduce the existence of a path isomorphism between A and B.

This counting argument has several unsatisfying aspects, which are caused by the implicit use of the following lemma.

Lemma 6 (Slicing of isomorphisms)**.** *Suppose $E' \subseteq E$ and $F' \subseteq F$ are finite sets, and that $f : E \to F$ and $g : E' \to F'$ are isomorphisms. Then there is an isomorphism $f \setminus g : E \setminus E' \to F \setminus F'$.*

The obvious proof of this lemma is by cardinality reasons. However this proof is, computationally speaking, "almost non-effective", in the sense that the isomorphism it produces implicitly depends on the choice of a total ordering for E and F. A consequence of that is that from any isomorphism in $\mathbf{Gam}$ we will extract an isomorphism in $\mathbf{Path}$, but we cannot hope its choice to be canonical, for any reasonable meaning of "canonical". Even worse, the witness isomorphisms given by this proof for $\simeq_k$ and $\simeq_{k+1}$ need not agree together. This implies that for infinitely deep arenas, one requires König's

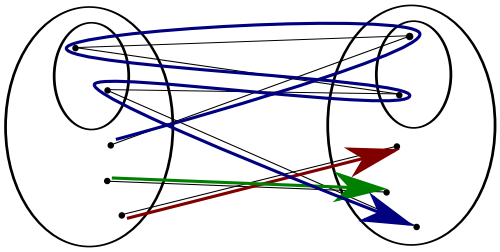


Figure 5. Slicing of isomorphisms.

lemma to actually build a path isomorphism from a game isomorphism. This means that we cannot deduce from the proof above an algorithm to extract path isomorphisms.

E. Extraction of a path isomorphism

To obtain a more computationally meaningful extraction of a path iso from a game iso, we must replace the proof of Lemma 6 by something else than counting. As formalized in the following proof, the idea is to remark that given the data of Lemma 6, starting from $x \in E \setminus E'$, the sequence

$$\begin{aligned} x_0 &= f(x) \\ x_{n+1} &= f \circ g^{-1}(x_n) \end{aligned}$$

must eventually reach $F \setminus F'$, as illustrated in Figure 5, yielding a bijection between $E \setminus E'$ and $F \setminus F'$.

PROPOSITION 2. *If $\phi : A \to B$ is a sequential play isomorphism, then for all $sa \in \mathcal{T}'_A$ with $\phi(sa) = \phi(s)b$, there is a family $(h^k_{s,sa})_{k\in\mathbb{N}}$ such that for all k, $h^k_{s,sa}$ is a k-isomorphism from a to b. This family is* coherent*, in the following sense: if $k_1 \leq k_2$, $h^{k_1}_{s,sa}$ is a prefix of $h^{k_2}_{s,sa}$.*

Proof: We will use the following notations. If $s \in \mathcal{T}'_A$, E_s will be the set of atomic extensions of s, that is of plays $sa \in \mathcal{T}'_A$, and F_s will be the set of atomic extensions of $\phi(s)$. For all plays $sa \in \mathcal{T}'_A$, although strictly speaking E_s is *not* a subset of E_{sa}, we have the following decomposition:

$$E_{sa} = E_s + J_a$$

Indeed, a move extending sa can either point to some s_i or to a. Note also that for any s, $\phi : sa \mapsto \phi(s)b$ induces an isomorphism $f_s : a \mapsto b$ from E_s to F_s.

For all $s \in \mathcal{T}'_A$ and $sa \in E_s$, we follow the reasoning illustrated in Figure 5 and consider a bipartite directed graph $G_{s,sa}$ defined as follows: its set of vertices is $V = E_{sa} + F_{sa}$ and its set of edges is $E = \{(x, f_{sa}(x)) \mid x \in E_{sa}\} + \{(y, f_s^{-1}(y)) \mid y \in F_s\}$. This graph is "deterministic", in the sense that the outwards degree of each vertex is at most one, moreover the only vertices whose outwards degree is 0 are those of J_b (where $b = f_s(a)$, so $F_{sa} = F_s + J_b$). Moreover $G_{s,sa}$ must be acyclic, since f_s and f_{sa} are isomorphisms. Thus from any vertex in J_a, there is an unique path in G leading to a vertex in J_b; this induces an isomorphism $g_{s,sa} : J_a \to J_b$. For each pair $(m, g_{s,sa}(m))$ we also keep track of the corresponding path $p^m_{s,sa} = (m, f_{sa}(m), f_s^{-1}(f_{sa}(m)), \ldots, g_{s,sa}(m))$.

It is now time to build the k-isomorphisms, by induction on k. For $k = 0$ this is obvious. For fixed $k + 1 \geq 1$, by induction hypothesis there is for each $sa \in \mathcal{T}'_A$ with $\phi(sa) = \phi(s)b$ a k-isomorphism $h^k_{s,sa}$ from a to b. In particular, for fixed $sa \in \mathcal{T}'_A$, consider the graph $G_{s,sa}$. Each of its edges of the form $(x, f_{sa}(x))$ are now labeled by the k-isomorphism $h^k_{sa,x}$ and all its edges of the form $(y, f_s^{-1}(y))$ are labeled by $(h^k_{s,f_s^{-1}(y)})^{-1}$. For each pair $(m, g_{s,sa}(m))$ we can now compose the labels along the path $p^m_{s,sa}$ and get a k-isomorphism $i_m : m \to g_{s,sa}(m)$. We then define $h^{k+1}_{s,sa} = (g_{s,sa}, (i_m)_{m\in J_a})$ which is as needed a $(k+1)$-isomorphism from a to b.

Note finally that if $k_1 \leq k_2$, $h^{k_1}_{s,sa}$ is a prefix of $h^{k_2}_{s,sa}$. This is proved by simultaneous induction on k_1 and k_2. If $k_1 = 0$ this is obvious. Otherwise, it relies on the fact that the graph $G_{s,sa}$ does not depend on k. Hence $h^{k_1+1}_{s,sa} = (g_{s,sa}, (i_m)_{m\in J_a})$ and $h^{k_2+1}_{s,sa} = (g_{s,sa}, (j_m)_{m\in J_a})$, and each i_m has be obtained from k_1-isomorphisms in the same way as j_m has been obtained from k_2-isomorphisms, so it immediately boils down to the induction hypothesis. ∎

THEOREM 1. *Two finitely branching arenas A and B are* $\mathbf{Gam}$*-isomorphic if and only if they are* $\mathbf{Path}$*-isomorphic.*

Proof: Consider an isomorphism $\sigma : A \Rightarrow B$ in $\mathbf{Gam}$. Restricted on plays with only two moves, it gives an isomorphism $f : I_A \to I_B$. By the previous proposition, there is for each $i \in I_A$ and for each $k \in \mathbb{N}$ a k-isomorphism $h^k_{\epsilon,i} : i \to f(i)$. Additionally, all these k-isomorphisms are compatible with each other, so they converge to an ω-isomorphism $h_{\epsilon,i} : i \to f(i)$. The iso f together with $h_{\epsilon,i}$ for all i define a path isomorphism from A to B. ∎

l) Canonicity: For each pair of arenas A, B, we have built a function $K_{A,B} : \mathbf{Gam}_i(A, B) \to \mathbf{Path}_i(A, B)$. The natural question is then whether, like in the other cases, this function defines a full functor. Unfortunately the answer is no, in fact $K_{A,B}$ is not even functorial. Indeed, the construction is based on the more explicit proof of Lemma 6 illustrated in Figure 5, which is not functorial; one can find sets $E' \subseteq E$, $F' \subseteq F$ and $G' \subseteq G$ and isomorphisms $E \xrightarrow{f} F \xrightarrow{g} G$ and $E' \xrightarrow{f'} F' \xrightarrow{g'} G'$ such that $(f \setminus f'); (g \setminus g') \neq (f; g) \setminus (f'; g')$. From this it is not hard to find a counter-example to the functoriality of $K_{A,B}$. It is a bit lengthy to describe it properly though, so we do not include it.

Although not being a functor, K does satisfy some canonicity property: its result is invariant under renaming of moves in A and B. In other terms, $K_{A,B} : \mathbf{Gam}_i(A, B) \to \mathbf{Path}_i(A, B)$ is natural in A and B, if both $\mathbf{Gam}_i(-,-)$ and $\mathbf{Path}_i(-,-)$ are seen as bifunctors from $\mathbf{Path}_i^{op} \times \mathbf{Path}_i$ to $\mathbf{Set}$ (using implicitly the faithful functor from $\mathbf{Path}_i$ to $\mathbf{Gam}_i$ of Figure 4).

F. Application to $\mathcal{L}_2$

Our isomorphism theorem most naturally applies to $\mathbf{Gam}$ (so to call-by-name languages), but $\mathcal{L}_2$ is modeled in $\mathrm{Fam_f}(\mathbf{Gam})$, and more precisely in the Kleisli category of the strong monad T, so we have to check how our result extends to this. Let us first relate isomorphisms in $\mathrm{Fam_f}(\mathbf{Gam})_T$ and isomorphisms in $\mathbf{Gam}$ using the following lemma.

LEMMA 7. *Let $A = (A_i)_{i\in I}$ and $B = (B_j)_{j\in J}$, then isomorphisms between A and B in $\mathrm{Fam_f}(\mathbf{Gam})_T$ are in one-to-one correspondence with pairs $(f, (\sigma_i)_{i\in I})$ where $f : I \to J$ is a bijection and each $\sigma_i : A_i \Rightarrow B_{f(i)}$ is an iso in $\mathbf{Gam}$.*

Proof: Imagine $\sigma : A \to T(B)$ and $\tau : B \to T(A)$ are morphisms in $\mathrm{Fam_f}(\mathbf{Gam})$ that form an isomorphism in $\mathrm{Fam_f}(\mathbf{Gam})_T$. Here A and B are families of arenas, so $A = (A_i)_{i\in I}$, and $\sigma = (\sigma_i)_{i\in I}$ with $\sigma_i : A_i \Rightarrow T(B)$. Similarly, we have $B = (B_j)_{j\in J}$ and $\tau_j : B_j \Rightarrow T(A)$. Then, first note that σ_i and τ_j necessarily first give an answer to the initial Opponent move in T, *i.e.* the initial question of the lifted sum in $T(B)$ and $T(A)$. Indeed take $i \in I$, and consider $\sigma_i; \tau^* : A_i \to T(A)$, where $\tau^* : T(B) \to T(A)$ is the *lifting* of $\tau : B \to T(A)$. This morphism must be a component of the identity on A in $\mathrm{Fam_f}(\mathbf{Gam})_T$ since σ, τ form an iso. In particular, it does directly answer the initial move in T. However, by definition of $\tau^* : T(B) \to T(A)$ it is *strict*, *i.e.* it directly interrogates the left occurrence of T, so σ_i must necessarily first answer the initial move in T otherwise we would immediately get a contradiction.

This means that each σ_i must first choose a component $j \in J$ (thus inducing a function $f : I \to J$), then play as $\sigma'_i : A_i \Rightarrow B_j$. The same analysis on τ provides a function $g : J \to I$ and strategies $\tau'_j : B_j \Rightarrow A_{g(j)}$, and it is then obvious that since σ, τ form an isomorphism g must be inverse of f and each τ'_j inverse of $\sigma'_{g(j)}$. ∎

m) Syntactic characterization: Let us prove now that the equational theory $\mathcal{E}$ given in Figure 1 characterizes the types A and B such that $[\![A]\!]$ and $[\![B]\!]$ are isomorphic in $\mathrm{Fam_f}(\mathbf{Gam})_T$.

LEMMA 8 (Type normal form)**.** *Any type A has an unique representative (up to $\simeq_{\mathcal{E}}$) generated by the non-terminal S in:*

$$\begin{array}{rcl} S & ::= & \mathtt{bool}^n \times T \\ T & ::= & \mathtt{unit} \mid \Pi_{i\in I} U \\ U & ::= & T \to S \end{array}$$

Proof: Straightforward. ∎

PROPOSITION 3. *If $[\![A]\!]$ and $[\![B]\!]$ are isomorphic in $\mathrm{Fam_f}(\mathbf{Gam})_T$, then $A \simeq_{\mathcal{E}} B$.*

Proof: By induction on their normal forms. For types generated by S take $A \simeq_{\mathcal{E}} \mathtt{bool}^n \times A'$ and $B \simeq_{\mathcal{E}} \mathtt{bool}^p \times B'$. By Lemma 7 we have $n = p$ (since $[\![A']\!]$ and $[\![B']\!]$, generated by T, must be singletons) and we still have $[\![A']\!] \simeq_{\mathrm{Fam_f}(\mathbf{Gam})_T} [\![B']\!]$. The case of types generated by T and U is direct. ∎

THEOREM 2. *For any types A, B of $\mathcal{L}_2$ whose interpretation give families $(A_i)_{i\in I}$ and $(B_j)_{j\in J}$ the following propositions are equivalent:*

(1) $A \simeq_{\mathcal{L}_2} B$
(2) $(A_i)_{i\in I} \simeq_{\mathrm{Fam_f}(\mathbf{Gam})_T/\cong} (B_j)_{j\in J}$
(3) $(A_i)_{i\in I} \simeq_{\mathrm{Fam_f}(\mathbf{Gam})_T} (B_j)_{j\in J}$
(4) $A \simeq_{\mathcal{E}} B$

Proof: (1) ⇒ (2) by soundness and by definition of type isomorphisms in $\mathcal{L}_2$, (2) ⇒ (3) because the arenas are complete, hence every play of the identity is a prefix of a complete play of the identity, so any isomorphism $\sigma : A \Rightarrow A$ such that $\mathrm{comp}(\sigma) = \mathrm{comp}(id_A)$ must satisfy $id_A \subseteq \sigma$. But as isomorphisms both are total strategies, so $\sigma = id_A$. (3) ⇒ (4) by Proposition 3. Finally, (4) ⇒ (1) because equations in $\mathcal{E}$ can be implemented in the syntax of $\mathcal{L}_2$. ∎

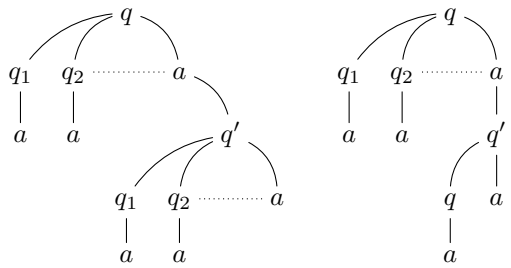


Figure 6. Non-trivial isomorphic arenas in $\mathbf{Gam}_\infty$

G. Isomorphisms in the presence of nat

As suggested by the importance of counting in the proof, the presence of nat makes it possible to build a non-trivial isomorphism by playing Hilbert's hotel. Consider the programming language $\mathcal{L}$ from [1], obtained from $\mathcal{L}_2$ by replacing bool with nat. This language has a fully abstract interpretation in $\mathrm{Fam}(\mathbf{Gam}_\infty)_T$, where $\mathbf{Gam}_\infty$ is the category of not necessarily finitely branching arenas, and single-threaded strategies.

PROPOSITION 4. *There is an isomorphism in $\mathrm{Fam}(\mathbf{Gam}_\infty)$ between $[\![(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{unit}) \to \mathtt{unit}]\!]$ and $[\![(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{unit}) \to \mathtt{unit}]\!]$.*

Proof: By definition of the interpretation of types, this boils down to an isomorphism in $\mathbf{Gam}_\infty$ between the two arenas represented in Figure 6. Informally, the isomorphism from left to right can be described as follows. As long as q' has not been played, it behaves as the identity. Whenever Opponent plays q', it copies it to the other side. Then if q' has only appeared once, there are two available copies of $\mathtt{nat} \to \mathtt{unit}$ on the left side, one $\mathtt{nat} \to \mathtt{unit}$ and one $\mathtt{unit} \to \mathtt{unit}$ on the right side, so Player picks a bijection between $\mathbb{N} + \mathbb{N}$ and $\mathbb{N} + 1$ and plays accordingly. More generally, if q' has appeared n times, there are exactly $n + 1$ available copies of $\mathtt{nat} \to \mathtt{unit}$ on the left hand side, one copy of $\mathtt{nat} \to \mathtt{unit}$ and n copies of $\mathtt{unit} \to \mathtt{unit}$ on the right hand side, so Player has to follow a bijection between $(n+1)\mathbb{N}$ and $\mathbb{N}+n$. So any choice of a bijection between $(n+1)\mathbb{N}$ and $\mathbb{N}+n$ (for all $n \in \mathbb{N}$) will provide an isomorphism. ∎

These strategies are not compact so the definability theorem does not apply, however we can nonetheless manually extract corresponding programs from them. We display them in Figure 7, where $*$ denotes the product operation on natural numbers, and $\mathtt{div}\ M\ N$ outputs the result of the division algorithm on $M : \mathtt{nat}$ and $N : \mathtt{nat}$. Unfortunately, these terms are too complex to hope for a reasonably-sized direct proof that their interpretations give the strategies described above or even that they form an isomorphism. This kind of difficulty emphasizes the need for new algebraic methods to manipulate and prove properties of imperative higher-order programs.

```
f : (nat → unit) → (nat → unit) → unit ⊢
 new count := 0, func := ⊥ in
  λg : nat → unit.
   let x = f (λn. g(n ∗ (!count + 1))) in
    λh : unit → unit.
     count := !count + 1;
     let c = !count in
      func := let h = !func in λn. if n = !count then h else g n;
      x (λn. if n = 0 then !func c ()
                      else g((n − 1) ∗ (!count + 1) + c))
```

```
f : (nat → unit) → (unit → unit) → unit ⊢
 new count := 0, func := ⊥ in
  λg : nat → unit.
   let x = f (λn.
    let (q, r) = div n (!count + 1) in
     if r = 0 then g q else !func r (q + 1))
   in
    λh : nat → unit.
     count :=!count + 1;
     func := let h = !func in λn. if n = !count then h else h n;
     let c =!count in x (λ_. !func c 0)
```

Figure 7. Type isomorphism in $\mathcal{L}$ between $(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{unit}) \to \mathtt{unit}$ and $(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{unit}) \to \mathtt{unit}$.

IV. Conclusion

We solved Laurent's conjecture and characterized the isomorphisms of types in $\mathcal{L}_2$. Surprisingly, we realized that the combination of higher-order references, natural numbers and call-by-value allowed to define new non-trivial type isomorphisms. Note however that if well-bracketing is satisfied, the proof of our core game-theoretic theorem adapts directly to arenas where all moves only enable a finite number of questions, but an arbitrary numbers of answers. As a consequence, there are no non-trivial isomorphisms (*i.e.* not already present in the λ-calculus) in the call-by-name variant of $\mathcal{L}$, although we can define one using `call/cc`.

Note that despite the seemingly restricted power of $\mathcal{L}_2$, our theorem does apply to all real-life programming languages that have a bounded type of integer, such as $\mathtt{bool}^{32}$ or $\mathtt{bool}^{64}$: in this setting, no non-trivial isomorphism can exist. However unbounded natural numbers can be defined using recursive types, so the isomorphism above can be implemented in a call-by-value programming language with recursive types and general references, such as OCaml.

This work can be extended in several different ways. An obvious possibility is to study isomorphisms in the presence of sum types, since the model is already equipped to handle them. We could also try to eliminate bad variables. Murawski and Tzevelekos' games model [22] of Reduced ML may be a good setting to try that, however it is not clear whether our core results can be reproved in their nominal setting.

Acknowledgments. We would like to thank Guy McCusker for interesting discussions on the games models for state and for his help to get a term from the strategy described in Proposition 4, and the anonymous referees for their useful comments and suggestions. We also would like to acknowledge the support of (UK) EPSRC grant RC-CM1025.

References

[1] Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *13th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1998.

[2] Samson Abramsky and Guy McCusker. Linearity, Sharing and State: a Fully Abstract Game Semantics for Idealized Algol with active expressions, 1997.

[3] Samson Abramsky and Guy McCusker. Call-by-value games. In Mogens Nielsen and Wolfgang Thomas, editors, *6th Annual Conference of the European Association for Computer Science Logic*, volume 1414 of *Lecture Notes in Computer Science*. Springer, 1998.

[4] Frank Atanassow and Johan Jeuring. Inferring type isomorphisms generically. In Dexter Kozen and Carron Shankland, editors, *MPC*, volume 3125 of *Lecture Notes in Computer Science*, pages 32–53. Springer, 2004.

[5] Gilles Barthe and Olivier Pons. Type isomorphisms and proof reuse in dependent type theory. In Furio Honsell and Marino Miculan, editors, *FoSSaCS*, volume 2030 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2001.

[6] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[7] Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Longo. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 2(2):231–247, 1992.

[8] Kim B. Bruce and Giuseppe Longo. Provable isomorphisms and domain equations in models of typed languages (preliminary version). In *STOC*, pages 263–272. ACM, 1985.

[9] M. Dezani-Ciancaglini. Characterization of normal forms possessing inverse in the λ-β-η-calculus. *Theoretical Computer Science*, 2(3):323–337, 1976.

[10] R. Di Cosmo. Invertibility of terms and valid isomorphisms, a proof theoretic study on second order lambda-calculus with surjective pairing and terminal object. Technical report, Technical Report TR 10-91, LIENS Ecole Normale Supérieure, Paris, 1991.

[11] Marcelo P. Fiore, Roberto Di Cosmo, and Vincent Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. *Ann. Pure Appl. Logic*, 141(1-2):35–50, 2006.

[12] Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1-2):393–456, 1999.

[13] Martin Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163(2):285–408, December 2000.

[14] Martin Hyland and Andrea Schalk. Games on graphs and sequentially realizable functionals. In *Logic in Computer Science 02*, pages 257–264, Kopenhavn, July 2002. IEEE Computer Society Press.

[15] James Laird. Full abstraction for functional languages with control. In *12th IEEE Symposium on Logic in Computer Science*, pages 58–67, 1997.

[16] James Laird. A game semantics of the asynchronous π-calculus. In *CONCUR*, pages 51–65, 2005.

[17] Olivier Laurent. Polarized games. *Annals of Pure and Applied Logic*, 130(1-3):79–123, 2004.

[18] Olivier Laurent. Classical isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):969–1004, 2005.

[19] Guy McCusker. Games and full abstraction for a functional metalanguage with recursive types. PhD thesis, Imperial College, University of London, 1996. Published in Springer-Verlag's Distinguished Dissertations in Computer Science series, 1998.

[20] Paul-André Melliès and Nicolas Tabareau. An algebraic account of references in game semantics. *Electr. Notes Theor. Comput. Sci.*, 249:377–405, 2009.

[21] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[22] Andrzej S. Murawski and Nikos Tzevelekos. Full Abstraction for Reduced ML. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2009.

[23] Mikael Rittri. Using types as search keys in function libraries. *J. Funct. Program.*, 1(1):71–89, 1991.

[24] Mikael Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *ITA*, 27(6):523–540, 1993.